Tiempo White Paper #3:
Introduction to Testing Tiempo Asynchronous Circuits

Version 1.0 - June 2, 2010

Overview

This document is a first introduction on Tiempo asynchronous circuits testing. It briefly presents the basics of Tiempo asynchronous logic that are required to understand the testing strategy proposed.

Outline

1 Introduction

Tiempo offers an innovative asynchronous and delay-insensitive design technology, with a portfolio of powerful asynchronous IP cores and a fully automated synthesis tool supporting these cores and their design technology.

Chips designed in this technology and/or integrating these asynchronous cores show outstanding physical properties such as ultra-low power, ultra-low noise, ability to work at ultra-low and variable voltage levels, reactivity (sleep mode by default, immediate wake-up), robustness against process-voltage-temperature (PVT) variations and resistance to hardware attacks (e.g., power analysis, fault injections).

The key reason for such capabilities is that Tiempo asynchronous circuits are fully clock-less and self-controlled, their behavior being governed by signal transitions rendez-vous and signal levels memorization. These principles, implemented with specific signal encoding and glitch-free logic, ensure functional correctness regardless of any actual delay through gates and wires. For more information on the fundamental design techniques used in Tiempo asynchronous circuits, the reader can refer to **Tiempo White Paper #1 "Tiempo Technology Introduction"**.

Tiempo synthesis tool – ACC (Asynchronous Circuit Compiler) – takes as input models written in the standard IEEE 1800 SystemVerilog language at transaction-level modeling (TLM), and generates as output standard Verilog gate-level netlists. These input models are written using a specific coding style as well as predefined SystemVerilog packages provided by Tiempo. For more information on the fundamental modeling techniques of Tiempo asynchronous circuits using SystemVerilog, the reader can refer to **Tiempo White Paper #2 "Introduction to SystemVerilog asynchronous Modeling"**.

The purpose of this document is to introduce the fundamentals of **Tiempo asynchronous circuits production testing**. It gives an understanding of the key properties inherent to Tiempo asynchronous circuits and suggests a smart revisiting and re-use of existing testing approaches developed for synchronous circuits.

2 Asynchronous circuits testing principles

2.1 Terminology

This section recalls the standard terminology used in the domain of VLSI circuit testing and defines the basic concepts in the context of asynchronous circuits testing.

2.1.1 Controllability and observability

Controllability and observability are defined with respect to the primary inputs and outputs of the circuit under test, i.e. the input and output ports of Tiempo's IPs.

The controllability of a circuit is the ability to set at "zero" or "one" any internal node of this circuit by means of applying given values on the primary inputs.

The observability of a circuit is the ability of deducing the value of any internal node of the circuit by recording the values available at the primary outputs.

2.1.2 Fault model and fault coverage

A fault is a physical defect introduced by the fabrication process that may lead to a failure, i.e. a behavior of the circuit that is different from the one specified.

Testing the circuit consists in determining whether the circuit includes faults. This is done by exercising the circuit using a set of test vectors applied to the primary inputs and recording the circuit's response at the primary outputs. Comparing the output values obtained during the test campaign with the expected output values pre-computed using the circuit reference enables to diagnose the presence of faults.

Quantifying the quality of the test campaign requires in a first step to precisely define the kind of faults that is targeted by the test (the "fault model"), and in a second step, given the fault model, to determine the parts of the circuit the test is able to reach as well as the parts of the circuits the test cannot cover (the "fault coverage").

A very common approach adopted when testing VLSI circuits is to model the physical defects at the logic gate level and consider the so-called "stuck-at fault" model. With this fault model it is assumed that a physical defect inside the circuit translates in the logical or gate domain into a node that is stuck-at one or stuck-at zero logically.

This model has been refined into two sub-models, the input stuck-at fault (considered here) and the output stuck-at fault models. Whereas the first one considers all the inputs of the gates of the circuit netlist, the second one, simpler, only considers the outputs of the gates (ignoring the wire branches of multiple fanout gates).

Hence, the test campaign consists in setting the primary inputs to control internal nodes and checking the primary outputs take the proper values. Unfortunately, it happens that some nodes of a given circuit may not be controllable or observable.

This is why the fault coverage is computed as the metric of test efficiency. The fault coverage is the percentage of faults that can be detected, i.e. the ratio of the faults that can be detected by the test sequence to the total number of possible faults in the circuit.

To appreciate the efficiency of the test method it is therefore important to consider the length of the test vector set, i.e. the number of values to apply to the primary inputs and the number of values to record when observing the primary outputs. For a given fault coverage, the smaller the test vector set, the better, and thus the faster the test campaign.

2.2 Tiempo asynchronous circuits properties for testing

Tiempo asynchronous logic has some key properties that are exploited to perform testing.

Tiempo asynchronous circuits are hazard-free, which means that there is no spurious transition inside the logic when processing a given input set. In other words, the circuit responds to the input, i.e converges to a stable state, by establishing its internal signals and outputs once, without exhibiting static or dynamic hazards on any gate output of the circuit.

Tiempo asynchronous circuits are delay insensitive. This property ensures that a transition at a primary input of a combinational block eventually causes an observable transition at a primary output of the block, and ensures that every gate switching in the block is necessary to compute at least one of the outputs.

This so-called “monotonic” behavior of the logic is exploited for stuck-at fault testing since a stuck-at prevents transitions to propagate properly inside the gate network, hence making the fault observable.

Tiempo asynchronous circuits are not controlled or sequenced using a clock but instead using local handshaking protocols (see Tiempo White Paper #1 “Tiempo Technology Introduction”).

Basically, Tiempo asynchronous circuits are built out of a collection of communicating modules. A module starts its computation when it receives a datum or a set of data. The data detection is achieved by the module exploiting a specific data encoding like the standard dual-rail encoding or other multi-rail encodings. As soon as the module delivers the result of its computation to another module or to an internal memory element, it acknowledges the inputs. It then receives a zero-data which actually corresponds to a return to zero phase of the protocol. Finally, the module answers by acknowledging the zero data which completes the communication protocol.

This computational scheme is actually implemented by a succession of wave-fronts that a stuck-at fault eventually stops. The fault then prevents the handshaking protocol to properly resume, a situation which can be observed.

Figure 1 illustrates a typical communication protocol using a dual-rail data encoding borrowed from Tiempo white paper #1.

Data	Invalid	0	1	Forbidden
D0	0	1	0	1
D1	0	0	1	1

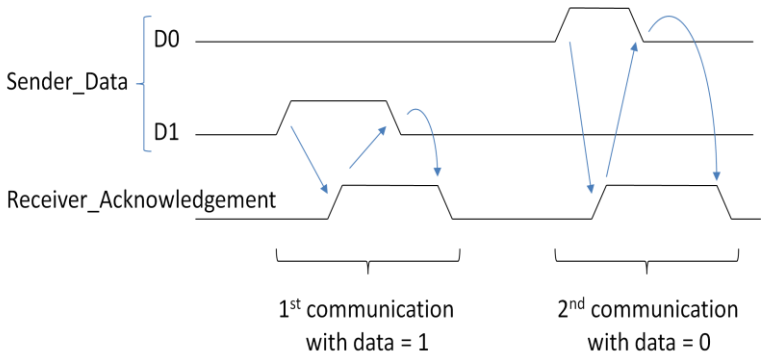

Figure 1: typical data encoding (dual-rail) and communication protocol.

It is clear from the diagram of Figure 1 that a stuck-at (zero or one) on one of the wire will eventually stop the handshaking protocols since every signal must switch from zero to one and back from one to zero.

3 Test strategy and coverage

Let's start by considering the circuit example of Figure 2, which is a simple four bit carry ripple adder. "FA" cells are full adders, "HA" is a half adder and "C" cells are Muller's C-Element gates. Data are dual rail encoded and initially all the nodes are at logic level zero.

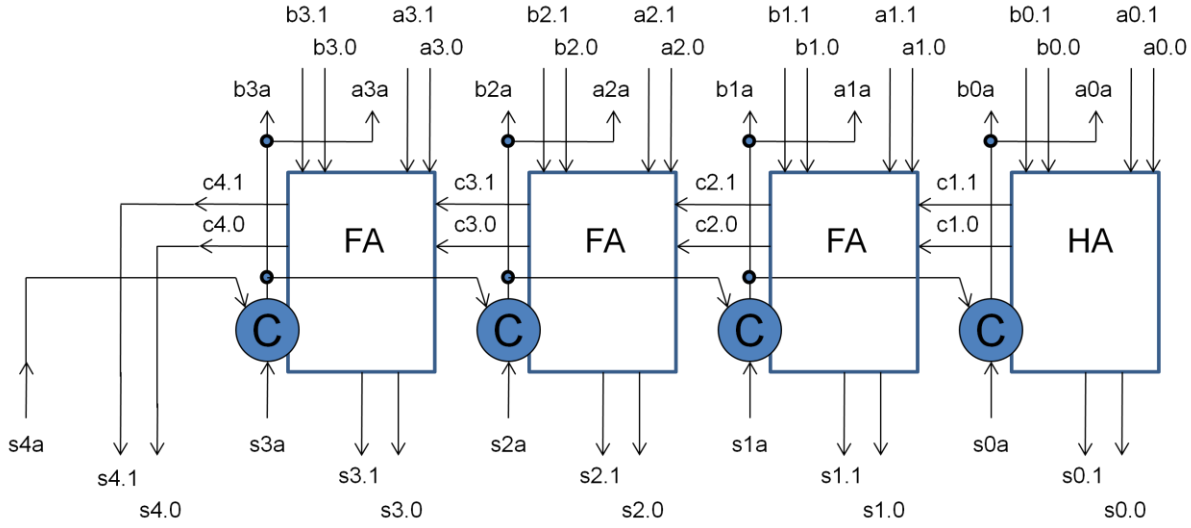

Figure 2 : circuit example.

Following the four phase handshaking protocol described in Figure 1, the computation starts when the input arrives which means that the two input bit vectors take a valid data value. The data wave front propagates into the gate network and produces a valid data at the output bit vector. The output acknowledges arrive and the acknowledgement wave front propagates backwards through the gate network to generate the input acknowledgement signals.

Now takes place the reset phase. The data reset wave front propagates to generate an invalid data at the outputs and follows the acknowledgement reset wave front which resets the input acknowledgement signals.

Figure 3 and Figure 4 illustrate the successive wave fronts that are taking place during the computation within the 4-bit adder. The input data are B={0,0,0,1} and A={1,1,1,1}.

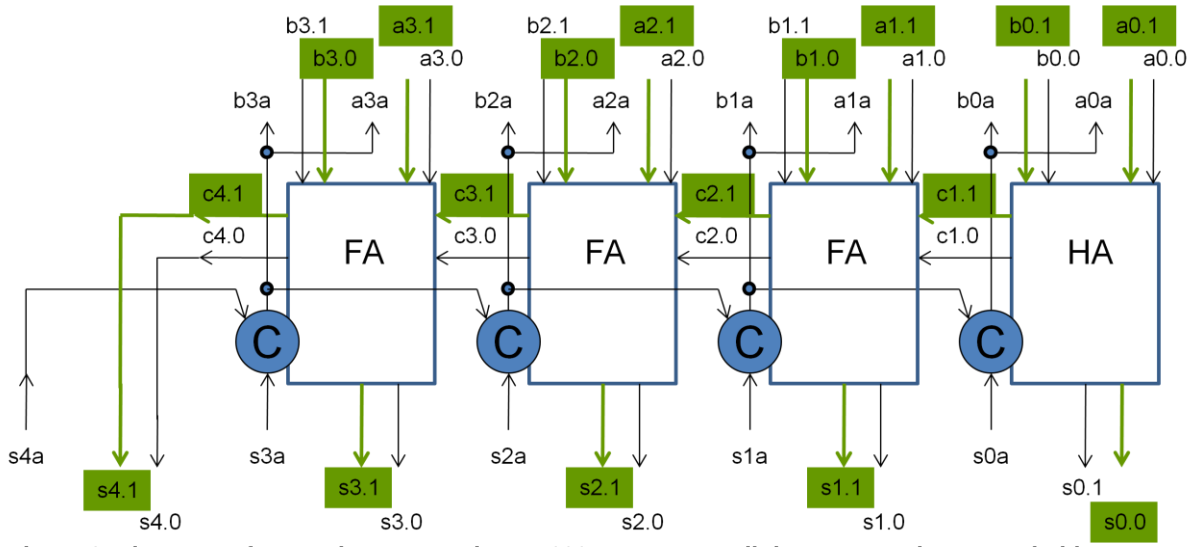

Figure 3 : data wave fronts when computing B=0001 + A=1111. All the green nodes are switching to one (data wave front) and then back to zero (data reset wave front).

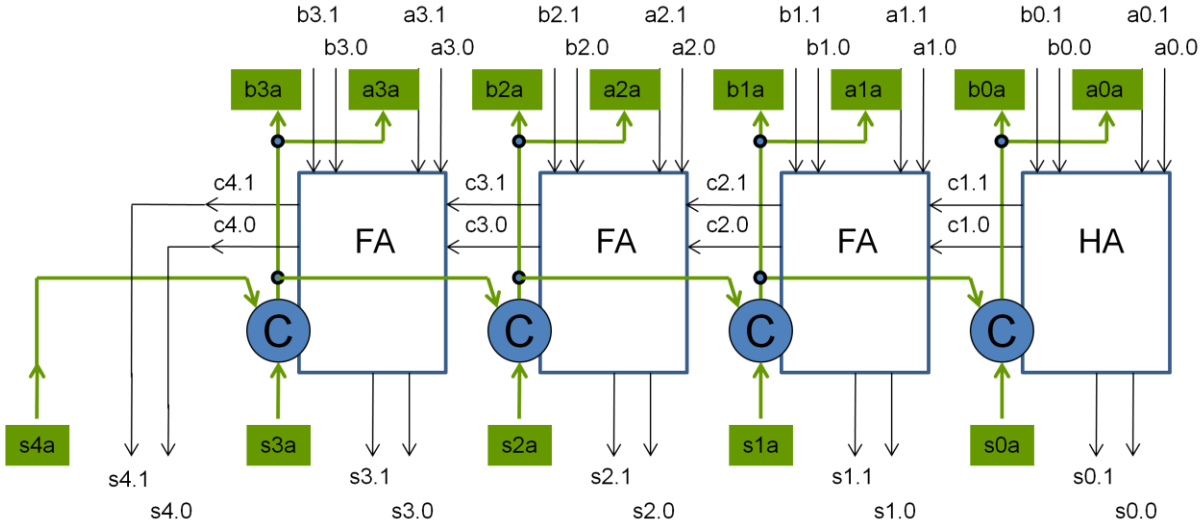

Figure 4 : acknowledgement wave front when computing B=0001 + A=1111. All the green nodes are switching to one (acknowledgement wave front) and then back to zero (acknowledgement reset wave front).

From this computation example, it is clear that an internal node participating to the computation switches to one and then switches back to zero: this is called a node “toggle”.

It is also clear that a node participating to the computation, either on the data path or on the acknowledgement path, is observable at one of the outputs at least. In fact,

each cell (FA, HA or C-Element) is designed so that transitions at all the outputs indicate that transitions occurred at all the inputs.

As a consequence, a stuck-at-0 fault will necessarily cause at least one of the outputs not to switch from zero to one, either a data or an acknowledgement. In fact, a stuck-at-0 fault actually stops the data or the acknowledgement wave front, preventing some of the outputs from switching.

Similarly, a stuck-at-1 fault will necessarily cause at least one of the outputs not to switch from one to zero (data or acknowledgement) since the data or acknowledgement reset wave front will be stopped, hence preventing one of the outputs from switching back to zero.

Therefore, the computation scheme described here-above has the following positive consequences on controllability and observability:

- a single computation always causes a subset of the internal nodes to toggle
- the subset of the internal nodes that toggle is solely determined by the input values and the gate network function (it does not depend on other parameters such as propagation time)
- a single or multiple stuck-at fault always causes one of the wave fronts involved in the computation to stop
- a single or multiple stuck-at fault always breaks the handshaking protocol which is easily observable.

These fundamentals properties are obviously exploited to test Tiempo asynchronous circuits.

3.1 Fault coverage

Testing such a gate network using the input stuck-at fault model requires applying a set of test vectors that exercises all the nodes of the circuit netlist to fully exercise the interconnections between the gates of the circuit.

In case of the four bit adder, computing 1111 + 0000, 0001 + 1111 and 0000 + 0001 exercises all the nodes of the netlist. Note that the number of test vectors remains the same whatever the length of the operands of this carry ripple adder: a 32 bit carry-ripple adder would also require three test vectors to exercise 100 % of the netlist nodes of Figure 2.

Indeed, Figure 5 illustrates the nodes exercised, i.e. toggling, for each test vector. Since the acknowledgement circuitry is actually tested three times the same way, Figure 5 reports four different couples (compute and reset) of wave fronts. There are three couples of data wave fronts corresponding to the three different input data values, and one couple of acknowledgement wave fronts which is common to all the input vectors.

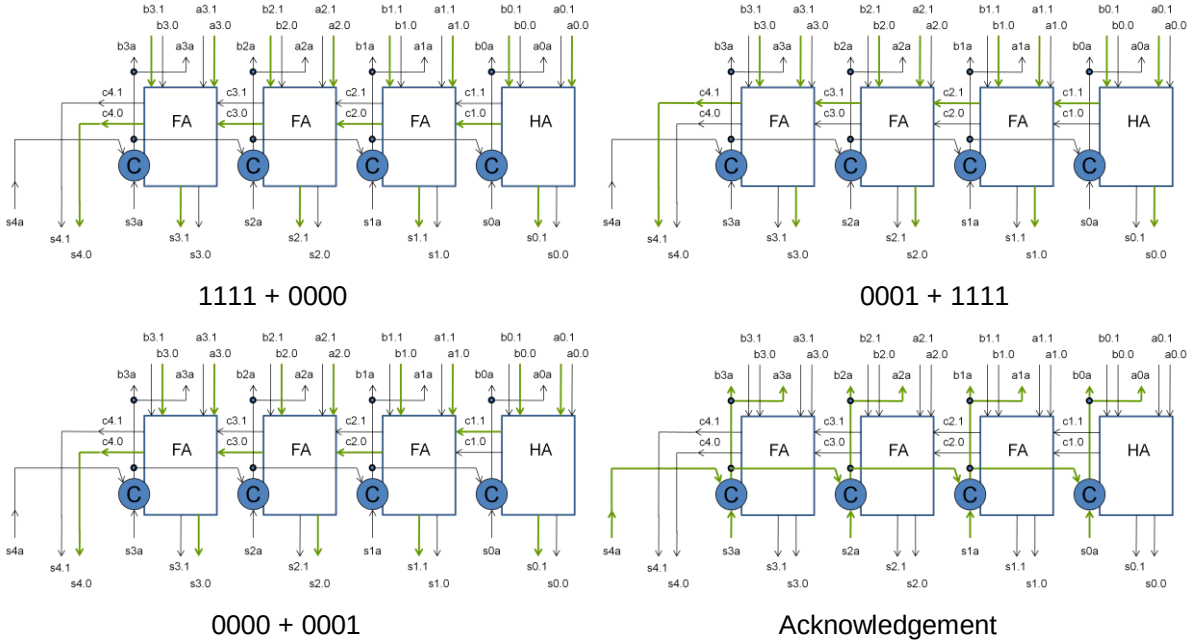

Figure 5 : 100 % fault coverage of the carry ripple adder using three test vectors under the input stuck-at fault model.

Computing the test coverage is then simply performed by counting the number of nodes which are toggling when running the test sequence, and dividing it by the total number of nodes present in the gate network. This is easily performed using a logic simulator able to report gate switching.

To extend the input stuck-at fault model, some other test vectors can be added to exercise the internal nodes of the gates and therefore improve even further the test quality. This strategy, which requires an accurate knowledge of the gate used, is similar to what is done when testing synchronous circuits and can also be applied. Note that, even with this strategy, the number of test vectors remains the same, whatever the length of the operands of the carry ripple adder. Also note that the number of test vectors required here is very small compared to the number of test vectors required for testing combinational circuits used in synchronous designs.

3.2 Test vector generation

Generating the test vectors to achieve 100% coverage of the netlist nodes can be formulated as a covering problem. In fact, it consists in finding the minimal set of input vectors that causes all the netlist nodes to toggle. As illustrated with a carry ripple adder, the input vector set is small and the number of test vectors can be independent of the width of the data path (in case of regular arithmetic operators).

4 Conclusion

This white paper introduced the basic principles for testing Tiempo asynchronous circuits. The test principles presented apply to any combinational functions and are extended when testing more complex circuits which include functions with memories.

Indeed, according to the storage elements used (pipeline latches, registers…), the test strategy is complemented in order to check the correctness of the data values. In some situations also, it is needed to improve the controllability of the design, especially in the presence of functional loops.

Anyway, it appears that a smart exploitation of Tiempo asynchronous circuit properties leads to an excellent trade-off between test-cost (Tiempo circuits controllability and observability), test-time (test at full speed and with small vector set) and test-coverage (Tiempo asynchronous logic properties) when applying functional-based test method rather than scan-based test method, even though Tiempo asynchronous circuits can be instrumented with scan chains.

In conclusion, the preferred test strategy which exploits Tiempo asynchronous circuits is based on functional testing as it is illustrated on a carry ripple adder in this white paper. The test mode consists in running a few set of computations ensuring to fully exercise the gate-level netlists under the gate-input stuck-at fault model. The presence of a fault is either detected by sensing that the test run does not complete (which in practice is easily implemented using a watch-dog timer), or by checking the incorrectness of the computation results.

For further information

Tiempo
110 rue Blaise Pascal
Bâtiment Viseo – Inovallée
38330 Montbonnot Saint Martin
FRANCE
T : +33 4 76 61 10 00
F : +33 4 76 44 19 69